Euclidean Algorithm (gcd) Time

Recall the core of the Euclidean algorithm for finding the greatest common divisor (gcd) of two integers `p` and `q`:

```
gcd(p, q):
    if q == 0:
        return p
    else:
        return gcd(q, p % q)
```

That is

$$\gcd(p, q) = \begin{cases} q & \text{if } q = 0 \\ \gcd(q, p \mod q) & \text{if } q \neq 0 \end{cases}$$

Euclidean Algorithm and Remainder Properties

The Euclidean algorithm is based on the principle that the gcd of two numbers also divides their difference. Specifically, if p and q are the numbers, then:
$\gcd(p, q) = \gcd(q, p \mod q)$

This means that each recursive step of the Euclidean algorithm replaces the larger number p with q and the smaller number q with the remainder $r = p \mod q$.

Key Property of Modulo Operation

When performing the modulo operation $r = p \mod q$, the remainder r satisfies:
$0 \leq r < q$

This is by definition of the modulo operation. Importantly, r is always strictly less than q.

Argument about Decrease by at Least a Factor of 2

To show that the second argument in the gcd function decreases by at least a factor of 2 for every second recursive call, we use the following reasoning:

1. **Initial Call:** $\gcd(p, q)$

   Here, q is the second argument.

2. **First Recursive Call:** $\gcd(q, r_1)$

   Where $r_1 = p \mod q$, so $0 \leq r_1 < q$.

3. **Second Recursive Call:** $\gcd(r_1, r_2)$

Where $r_2 = q \mod r_1$, so $0 \leq r_2 < r_1$.

Now, consider the relationship between q and r_2. Since $r_1 < q$, the value of r_1 could be anything from 0 to $q - 1$.

However, the crucial insight comes from the fact that:

$$r_2 = q \mod r_1$$

Since r_1 is now the divisor and q the dividend in the modulo operation, the remainder r_2 must be less than r_1. Importantly, we look at the case when r_1 is close to q, but since it can be no more than $q - 1$, for r_2 to be close to half of q, r_1 must be less than or equal to $q/2$.

In the worst-case scenario: $r_2 \leq q/2$

Thus, after every two recursive calls, the second argument (which starts as q) will be reduced to at most half of its original value.

Recursive Call Count Analysis

Let's now show that the gcd function uses at most $2\log_2 n + 1$ recursive calls, where n is the larger of p and q.

1. **Initial Call:** $\gcd(p, q)$

   Assume $p \geq q$, so $n = p$.

2. **After 2 calls:** The second argument has been reduced to at most $q/2$.

3. **After 4 calls:** The second argument has been reduced to at most $q/4$.

4. **Generalizing:** After $2k$ calls, the second argument is reduced to at most $q/2^k$.

The recursion stops when the second argument reaches 1 or 0. Therefore, the number of times you can halve q before it becomes less than or equal to 1 is $\log_2 q$. Since we are halving every two calls, the total number of calls required is:

$$2\log_2 q$$

Adding the initial call gives:

$$T(n) \leq 2\log_2 n + 1$$

where n is the larger of p and q.

Thus, the number of recursive calls $T(n)$ satisfies: $T(n) \leq 2\log_2 q + 1$

Since $q \leq p \leq n$, we can substitute q with n in the worst case, giving:
$T(n) \leq 2\log_2 n + 1$